Coles Scraper Documentation

Overview

This implementation of the Coles.com.au scraper is based on direct access to Coles' internal API endpoints. It avoids HTML parsing by using structured JSON data returned from Coles' backend. However, because these APIs are protected by Imperva Incapsula (a Web Application Firewall used by Coles), the scraper mimics real users through stealth mechanisms and manual CAPTCHA solving to prevent bot blocking and full data extraction.

This Coles.com.au scraper is built using Selenium Wire, Smartproxy, and MongoDB. It includes advanced anti-bot evasion techniques, such as rotating proxies, spoofing user agents, JavaScript-based stealth evasion (selenium-stealth), and realistic human-like interactions (e.g., delay, scrolls, click events). In comparison to the earlier approach of HTML parsing, using APIs to scrape data is more safe, scalable, and successful as it involves less interaction with a lot of UI elements.

The scraper is also built on ethical web scraping practices as we do not intend to cause harm to the website and use ethical practices to scrape data from publicly available sources. A detailed documentation on ethical scraping is also available on the documentations section.

Required Libraries and their Purpose

Library	Description	Purpose for this Project
selenium-wire	Extension of selenium	Captures requests and allows proxy configuration per session
selenium	Web automation library	Controls browser behavior programmatically
selenium-stealth	Anti-bot stealth utility	Masks WebDriver fingerprints and properties
requests	HTTP Client	Handles cookies and requests outside Selenium
pymongo	MongoDB client	Saves collected data into MongoDB.
configparser	INI file handler	Loads configuration values from configuration.ini

json,os,random,time,datetime	Core Python	Utilities for handling delays, paths, data writing

chromedriver – WebDriver for Chrome

chromedriver is an executable required to automate and control the Chrome browser via Selenium. We have used undetected chromedriver under the selenium wire library as a stealth mechanism to prevent detection and flagging by the web application firewall.

How it fits in the project:

```
import seleniumwire.undetected_chromedriver.v2 as uc
driver = uc.Chrome(
        seleniumwire_options=seleniumwire_options,
        options=options,
        version_main=135,
        use_subprocess=True
    )
```

Stealth and Anti-Bot Evasion Techniques

SmartProxy Proxies

Datacenter proxies are implemented from SmartProxy with location set to Australia. The current subscription gives us 100 IPs to use with 50 GB bandwidth.

Proxy set up code snippet:

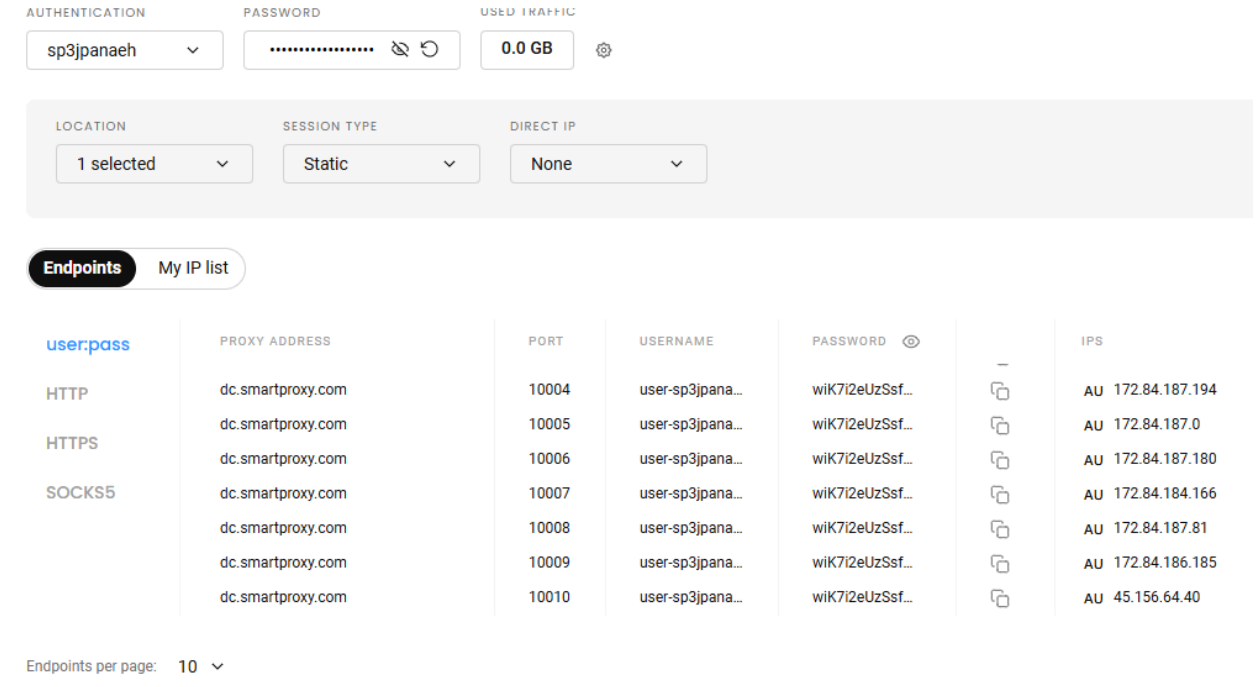
```
proxy_user_base = config.get('Smartproxy', 'Username')
    proxy_pass = config.get('Smartproxy', 'Password')
    proxy_host = config.get('Smartproxy', 'Host')
    ports_str = config.get('Smartproxy', 'Port')
    session_id = random.randint(100000, 999999)
    proxy_user = f"{proxy_user_base}-session-{session_id}"
    proxy_port = random.choice([port.strip() for port in
ports_str.split(',')])
    proxy_auth =
f"http://{proxy_user}:{proxy_pass}@{proxy_host}:{proxy_port}"

    seleniumwire_options = {
        'proxy': {
            'http': proxy_auth,
            'https': proxy_auth,
            'no_proxy': 'localhost,127.0.0.1'
        }
    }
```

Smartproxy details are obtained from the configuration.ini file.
IMP: You need to subscribe to Datacenter Proxy plan from SmartProxy with location set to Australia and enter your username, password, and related details below:-

```
[Smartproxy]
Username = your_username-au
Password = your_password
Host = dc.smartproxy.com
Port = 10001,10002,10003,10004,10005,10006,10007
```

SmartProxy Setup Dashboard with Location set to Australia



A random `session_id` is used to **rotate proxy identity**. Switches between proxy ports and rotates users to simulate different sessions.

User-Agent
The following user-agent is used in the automation process:

Code Snippet:

```
user_agent = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/135.0.0.0 Safari/537.36"
```

Browser fingerprint evasion via selenium-stealth

Code Snippet

```
stealth(driver,
    languages=["en-US", "en"],
    vendor="Google Inc.",
    platform="Win32",
    webgl_vendor="Intel Inc.",
    renderer="Intel Iris OpenGL Engine",
    fix_hairline=True,
    run_on_insecure_origins=True
)
```

Overrides key JavaScript-detectable values like navigator.language, navigator.vendor, navigator.platform, WebGLRenderingContext.vendor, and renderer

`fix_hairline=True` corrects rendering issues that bots typically cause.

Removes the `navigator.webdriver` flag via:

```
driver.execute_script("delete navigator.__proto__.webdriver")
```

Simulated Human Interaction

Code Snippet:

```
# This function sets random delay based on the parameters provided
def human_delay(min_sec=1.5, max_sec=3.5):
    if random.random() < 0.8:
        time.sleep(random.uniform(min_sec, max_sec))
```

Adds randomized delays to simulate a real user browsing.

CAPTCHA Handling

The captcha is manually handled now. Current logic detects if there is a captcha challenge and waits until the captcha is solved before continuing with the execution.

Code Snippet:

```
def is_captcha_present(driver):
    return "_incapsula_" in driver.current_url.lower() or ...
```

```
def wait_for_captcha(driver, timeout=120):
    try:
        print("CAPTCHA iframe detected. Waiting for it to disappear...")
```

```
        WebDriverWait(driver, timeout).until_not(
            EC.presence_of_element_located((By.XPATH,
"//iframe[contains(@src, '_Incapsula_Resource')]"))
        ) ...
```

It checks for _Incapsula_Resource iframes (used by Imperva). If captcha is found, it waits until we manually solve it.

Random Window Size

```
driver.set_window_size(random.randint(1000, 1400), random.randint(700,
1000))
```

The code above generates random window size every time a fresh session is created which mimics different devices.

Ethical Web Scraping

Humanized Delays

Humanized delays are added so that we do not place too much stress on the server.

Realistic User-Agents with rotation

Realistic user-agents are used.

Proxy Implementation and rotation

Proxy is implemented which can be rotated each session.

Coles API and Data Extraction Pattern

Coles Category Product API

This API is the main API we use to extract the data from Coles. It needs build_id, category_slug, and page number. For each category, we call this API by starting with page number 1 and looping until it returns no response. We continue this approach for all the categories until the data is fully extracted from all of them.

URL Pattern

```
https://www.coles.com.au/_next/data/{build_id}/en/browse/{category_slug}.js
on?page={page}&slug={category_slug}
```

build_id: A dynamic string extracted from the Coles webpage. It is extracted dynamically from the __NEXT_DATA__ script in the HTML now. Coles might change this pattern in the future so you might need to change the extraction logic accordingly.

category_slug: The SEO token for each category (e.g., meat-seafood, bakery).
page: The pagination number (starting from 1).

Note: Store ID and click and collect locations are set through the cookies which are sent with the requests. The products returned will be filtered based on this.

Response Parsing

```
{
  "pageProps": {
    "searchResults": {
      "results": [ ... products ... ]
    }
  }
}
```

Parsing Logic:

1. Check for pageProps → searchResults → results
2. Iterate through results where _type == "PRODUCT"
3. Extract product fields: id, name, pricing.now, pricing.was, pricing.comparable
4. Category from merchandiseHeir.category

Example Fields Parsed:

```
{
  "product_code": item.get("id"),
  "item_name": item.get("name"),
  "best_price": pricing.get("now"),
  ...
}
```

Coles Browse API (Category Discovery)

The coles browse API helps us to find what categories are available to browse. The categories change based on occasions and also can be available or not available based on a particular store and click and collect location. So, we use this API to find what categories are there for a particular store.

The C&C location and store ID are passed via cookies for this API as well.

URL Pattern:-

```
https://www.coles.com.au/_next/data/{build_id}/en/browse.json
```

Response Structure:

```
{
  "pageProps": {
    "allProductCategories": {
      "catalogGroupView": [
        {
          "level": 1,
          "type": "CATALOG",
          "name": "Bakery",
          "seoToken": "bakery",
          ...
        }
      ]
    }
  }
}
```

Parsing Logic:

1. Navigate to pageProps → allProductCategories → catalogGroupView
2. Filter only items with: level == 1, type == "CATALOG", productCount > 0
3. Store name and seoToken as a lookup dictionary for scraping.

BUILD ID Extraction (Required for all APIs)

The build_id is required as part of all _next/data/... API URLs. It changes frequently and is not static. So, we have used a logic to extract it dynamically from the webpage.

Source:

Embedded JSON object found in the HTML of the homepage:

```
<script id="__NEXT_DATA__" type="application/json">{"buildId": "abcd1234",
...}</script>
```

Parsing Logic:

```
json_data = driver.execute_script("return
document.getElementById('__NEXT_DATA__').textContent;")
build_id = json.loads(json_data).get("buildId")
```

If build_id is not found, scraping cannot proceed and the session is aborted. So, make sure to update the logic if Coles changes its patterns.

Current Limitations

The current scraping logic can fully scrape complete dataset from Coles. However, as Coles has protected their website with Incapsula, we do face CAPTCHA challenges occasionally and need to manually solve them 4-5 times throughout the full data extraction process.

Using high quality residential proxies did not result in CAPTCHA challenges and helped to do the complete run without any interruptions. This is because these proxies have good reputation which means less likely to get flagged. But, these proxies come with higher cost which makes it not feasible for an academic project with no budget.

So, datacenter proxies are used from SmartProxy which needs to be still paid but comes at a lower cost. But, it comes with challenges as well. Because these proxies are used by many people out there for scraping and other purposes, these IPs have a high chance that they are already flagged by web application firewalls. They also have a low reputation score. So, when we rotate proxies every time during our scraping run, we get a CAPTCHA challenge to solve at the beginning which needs to be solved manually. Hence, if we need to rotate 4-5 proxies during the complete run, we need to solve these challenges 4-5 times manually.

Future Work Areas

Based on the limitations above, there are a lot of areas we can work on in the future.

1. Collaborate with the machine learning team and develop an AI/ML model to solve CAPTCHAs automatically and integrate in the web scraping script.
2. Continuous maintenance is another area. Coles might change their APIs or patterns on how they return data. So, if something is changed, the scraper will be broken. In that case, we need to analyze their approach again on how they fetch and display their products on the front-end and modify the APIs and our parsing patterns again.
3. Research if there are any open source tools we can integrate to solve CAPTCHAs (especially hCaptcha as Incapsula uses hCaptcha) and try integrating it to solve CAPTCHAs.
4. Further modularize the code, do enhancements around error handling, logging mechanisms, and data validations.
5. Explore headless modes or ways to integrate the scraper into a server environment to automate the scraping process weekly.